Appendix A

Tag Propagation Model for various Verilog constructs

i) OPERATORS

Arithmetic

+	Propagate only if other operand is not 'X' or 'Z'
-	Propagate only if other operand is not 'X' or 'Z'
*	Propagate only if other operand is not '0', 'X' or 'Z'
/	Propagate only if other operand is not '0', 'X' or 'Z'
%	Propagate only if other operand is not '0', 'X' or 'Z'

Logical

!	Always propagate
&&	Propagate only if other operand is not '0', 'X' or 'Z'
||	Propagate only if other operand is '0'

Bitwise

~	Always propagate
&	For both scalars and vectors propagate only if i) Value of signal of interest is '0' and other operand is not '0' ii) Value of other operand is not '0', 'X' or 'Z'
|	For both scalars and vectors propagate only if i) Value of signal of interest is >= '1' and other operand is not >= '1' ii) Value of other operand is '0'
^	Propagate only if other operand is not 'X' or 'Z'
~^	Propagate only if other operand is not 'X' or 'Z'
^~	Propagate only if other operand is not 'X' or 'Z'

Reduction

&	Always propagate
~&	Always propagate
|	Always propagate
~|	Always propagate
^	Always propagate
~^	Always propagate
^~	Always propagate

Relational

>	Propagate only if other operand is not 'X' or 'Z'
>=	Propagate only if other operand is not 'X' or 'Z'
<	Propagate only if other operand is not 'X' or 'Z'
<=	Propagate only if other operand is not 'X' or 'Z'

Equality

==	Propagate only if other operand is not 'X' or 'Z'
!==	Propagate only if other operand is not 'X' or 'Z'
===	Always propagate
!===	Always propagate

Shift

<<	Left Operand : Propagate only if Right Operand is less than the size of the Left Operand and Right Operand is not 'X' or 'Z' Right Operand : Propagate only if Left Operand is not 'X' or 'Z'

>> Left Operand : Propagate only if Right Operand is less than the size of the Left Operand and Right Operand is not 'X' or 'Z'

Right Operand : Propagate only if Left Operand is not 'X' or 'Z'

Conditional

<condition> ? <true expression> : <false expression>

Always propagate any tags of variables of <condition> that observably determine whether condition is satisfied or not.
If <condition> is true, then propagate tag(s) corresponding to the <true expression>
If <condition> is false, then propagate tag(s) corresponding to the <false expression>

Concatenation

{} LHS : Inject tag on all signals on the LHS.
Propagate tags that are on RHS depending on operator type and variable values (See tag propagation for operators above).

RHS : Always Propagate

{{}} LHS : Inject tag on all signals on the LHS.
Propagate tags that are on RHS depending on operator type and variable values (See tag propagation for operators above).

RHS : Always Propagate

Special case : Concatenation on both LHS and RHS. For example,
{a,b,c} = {x,y};
In this case, we have to match the signals by size and propagate the tags associated with signals on the RHS to appropriate signals on the LHS.

Bit Select :

Part Select :

LHS : We associate tags with the corresponding bit (instead of the whole variable). Tag injection/propagation happens based on bits.

RHS : If there are tags associated with bits (because of an earlier use of bit or part select of the variable in LHS context), then tags corresponding to the bits are considered. Otherwise, tags

associated with the variable as a whole is considered.

ii) STRUCTURAL CONSTRUCTS

Module Definition : No action

UDP Definition : No action

Continuous Assignment :

Inject tag on LHS.
Propagate tags that are on RHS depending on operator type and variable values
(See tag propagation for operators above).

Gate Instantiation :

Inject tag on gate output.
Propagate tags that are on the inputs, depending on gate type and input values, as follows:

Gate Type	Action
buf	Always Propagate
not	Always Propagate
and	Same as binary bitwise &
or	Same as binary bitwise |
xor	Same as binary bitwise ^
nand	Same as binary bitwise &
nor	Same as binary bitwise |
xnor	Same as binary bitwise ^
bufif0	Control input : Always Propagate Data input : Propagate only if Control input is '0'
bufif1	Control input : Always Propagate Data input : Propagate only if Control input is '1'
notif0	Control input : Always Propagate Data input : Propagate only if Control input is '0'

notif1	Control input : Always Propagate Data input : Propagate only if Control input is '1'
tran	Always Propagate
tranif0	Control input : Always Propagate Data inputs : Propagate only if Control input is '0'
tranif1	Control input : Always Propagate Data inputs : Propagate only if Control input is '1'
rtran	Always Propagate
rtranif0	Control input : Always Propagate Data inputs : Propagate only if Control input is '0'
rtranif1	Control input : Always Propagate Data inputs : Propagate only if Control input is '1'
nmos	Control input : Always Propagate Data input : Propagate only if Control input is '0'
pmos	Control input : Always Propagate Data input : Propagate only if Control input is '1'
cmos	Control input : Always Propagate Data input : Propagate only if Control input is '0' or '1'
rnmos	Control input : Always Propagate Data input : Propagate only if Control input is '0'
rpmos	Control input : Always Propagate Data input : Propagate only if Control input is '1'
rcmos	Control input : Always Propagate Data input : Propagate only if Control input is '0' or '1'
pullup	No action
pulldown	No action

Module instantiation :

Propagate tags that are on the variables connected to the inputs to corresponding module inputs. Propagate tags that are on the module outputs to the corresponding variables connected to them.

When there is a complex expression for the terminal connection for an input port, tags will be propagated depending on operator type and signal values (See tag propagation for operators above).

For inouts, propagate in both directions.

UDP Instantiation : Always propagate

Specify Block : No action
---

iii) BEHAVIORAL CONSTRUCTS

Procedural Assignments :

Blocking Assignment (=)

Inject tag on LHS.
Propagate tags that are on RHS depending on operator type and variable values.
(See tag propagation for operators above).

Non-Blocking Assignment (<=)

Inject tag on LHS.
Propagate tags that are on RHS depending on operator type and variable values.
(See tag propagation for operators above).
This propagation should be scheduled to be done at the end of the timestep.

Procedural Continuous Assignments :
---

assign : Inject tag on LHS.
Propagate tags that are on RHS depending on operator type and variable values.
(See tag propagation for operators above).

deassign : Ignore

force : Ignore

release : Ignore

Conditional Statements :
---

if-else-if

```
if (condExp)
begin
  stmt1;
  stmt2;
end
else
```

```
begin
  stmt3;
  stmt4;
end
```

If the condition expression (condExp) is a named variable, then propagate the tags associated with it to the LHS of the statements executed within the if-branch if condExp is true (stmt1 and stmt2 in this example) or the statements executed within the else-branch if condExp is false (stmt3 and stmt4 in this example).

If the condition expression (condExp) is a complex expression, then the decision regarding which tags associated with the variables in condExp are propagated is taken depending on the operator type and signal values (See tag propagation for operators above).

Also propagate tags of the RHS observably effecting variables of stmt1 and stmt2 (if condExp is true) or stmt3 and stmt4 (if condExp is false).

case

```
case (caseExpr)
 Item1:
 begin
    stmt1;
    stmt2;
 end
 Item2:
 begin
    stmt1;
    stmt2;
 end
 default:
 begin
    stmt1;
    stmt2;
 end
endcase
```

If the condition expression (caseExpr) is a named variable, then propagate the tags associated with it to the LHS of the statements executed within the matching case item.

If the condition expression (caseExpr) is a complex expression, then the decision regarding which tags associated with the variables in condExp are propagated is taken depending on the operator type and signal values (See tag propagation for operators above).

Also propagate tags of the observably effecting RHS variables of assignments corresponding to the matching case item.

casex : Same as above
---

casez : Same as above

Looping statements :
---

forever : No action.
Tag injection and propagation happens for the statements within the forever loop as usual.

repeat . : No action.
Tag injection and propagation happens for the statements within the repeat loop as usual.

while : If the condition expression is a named variable, then propagate the tags associated with it to the LHS of the statements executed within the while loop. If the condition expression is a complex expression, then the decision regarding which tags associated with the variables in condExp are propagated is taken depending on the operator type and signal values (See tag propagation for operators above).

Also propagate tags on the RHS observably-effecting variables of assignments within the while loop.

for : No action.
Ignore the for loop index.
Tag injection and propagation happens for the statements within the for loop as usual.

Procedural Timing control :

Delay Control (#) : No action

Event Control :

@ Propagate only if change has occurred

@posedge Propagate only if there is a posedge on the signal

@negedge Propagate only if there is a negedge on the signal

wait Propagate only if change has occurred

For the above four cases, we propagate to the LHS of all the following statements.

Event or Operator :
---

or Propagate only if change has occurred in the signal of interest and not on the other.

We propagate to the LHS of all of the or-triggered assignment statements.

Example:

```
always @(a or b)
begin
  x = ~y;
  z = y;
end
```

If "a" is the signal that has changed, then the tag of "a" will be transferred to "x" and "z" (LHS of or-triggered assignment statements). The other signal "b," which did not change, will not have its tags propagated.

Named event :

-> No action

Named Blocks :

Sequential Block (begin - end) : No action

Parallel Block (fork - join) : No action

Always : No action

Task Definition : No action

Task Enable :

Since Verilog tasks are not reentrant (unlike C functions), we do not need to do any save-restore sort of action. When a task is enabled, the tags associated with the actual input parameters are propagated to the formal input parameters. Tags in global signals accessed within the task are maintained as they are. When the task exits, the tags that are associated with the formal outputs of the task are propagated to the actual output parameters of the task enable.

Function Definition : No action

Function Call :

Since Verilog functions are not reentrant (unlike C functions), we do not need to do any save-restore sort of action. When a function is called, the tags associated with the actual input parameters are propagated to the formal input parameters. Tags in global signals accessed within the function are maintained as they are. When the function exits, the tags that are associated with the function are propagated to the temporary variable which holds the function return value. These tags are then propagated to the LHS of the statement depending on operator type and signal values (See tag propagation for operators above).

Disable : No action
